

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problems Mailbox.**

CLIPPEDIMAGE= JP02001034619A

PUB-NO: JP02001034619A

DOCUMENT-IDENTIFIER: JP 2001034619 A

TITLE: STORE AND RETRIEVAL METHOD OF XML DATA, AND XML DATA RETRIEVAL SYSTEM

PUBN-DATE: February 9, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
------	---------

KANEMASA, YASUHIKO	N/A
--------------------	-----

KUBOTA, KAZUMI	N/A
----------------	-----

ISHIKAWA, HIROSHI	N/A
-------------------	-----

INT-CL\_(IPC): G06F017/30

ABSTRACT:

PROBLEM TO BE SOLVED: To make storable XML data into a data base and to make executable a complicated inquiry at a high speed.

SOLUTION: A relation data base of an XML data store means 1 includes an intermediate node table 2 which stores the intermediate node information, a link table 3 which stores the link information, a leaf node table 4 which stores the leaf nodes, an attribute table 5 which stores the attribute information, a path ID table 6 where the path IDs are made to correspond to the character strings and a label ID table 7 where the label IDs are made to correspond to the character strings. The XML data which are expressed in a tree structure are divided into nodes, and these nodes are made to correspond to the link information and stored in the tables 2-7. When the XML data are retrieved, an inquiry statement is given to an inquiry processing means 9. The means 9 executes an inquiry to track a tree structure by using index 8 and outputs a requested retrieval result.

COPYRIGHT: (C)2001,JPO

DERWENT-ACC-NO: 2001-230893

DERWENT-WEEK: 200124

4~COPYRIGHT 1999 DERWENT INFORMATION LTD 14~

BASIC-ABSTRACT: NOVELTY - The tree structure of extensible mark-up language (XML) is divided into nodes and links. Information contained in intermediate and branch nodes along with links are collected and stored in the tables (2-4) in relational database memory (1). XML data with tree structure is searched with reference to the tables.

ADVANTAGE - Since the tree structure of XML is divided into nodes and links, the searching of XML data is done at a high speed even if the data structure is unique.

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-34619

(P2001-34619A)

(43) 公開日 平成13年2月9日(2001.2.9)

(51) Int.Cl.

識別記号

F I

テーマコード(参考)

G 0 6 F 17/30

G 0 6 F 15/419

3 2 0

5 B 0 7 5

15/403

3 3 0 B

3 4 0 D

審査請求 未請求 請求項の数 5 O L (全 15 頁)

(21) 出願番号

特願平11-203908

(22) 出願日

平成11年7月16日(1999.7.16)

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中4丁目1番  
1号

(72) 発明者 金政 泰彦

神奈川県川崎市中原区上小田中4丁目1番  
1号 富士通株式会社内

(72) 発明者 久保田 和己

神奈川県川崎市中原区上小田中4丁目1番  
1号 富士通株式会社内

(74) 代理人 100100930

弁理士 長澤 俊一郎 (外1名)

最終頁に続く

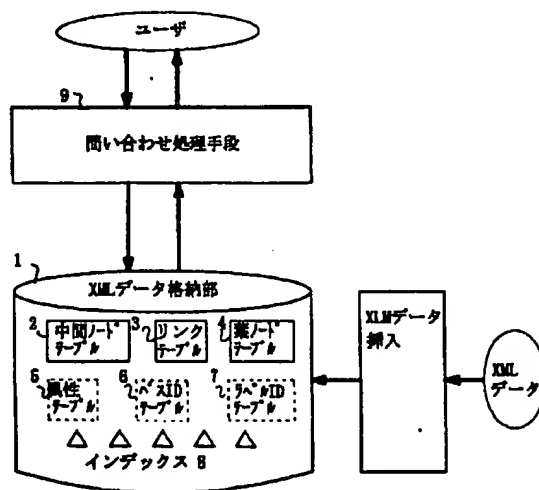
(54) 【発明の名称】 XMLデータの格納/検索方法およびXMLデータ検索システム

(57) 【要約】

【課題】 XMLデータをデータベースに格納し、複雑な問い合わせを高速に実行できるようにすること。

【解決手段】 XMLデータ格納手段1の関係データベースに、中間ノードの情報を格納する中間ノードテーブル2、リンクの情報を格納するリンクテーブル3、葉ノードの情報を格納する葉ノードテーブル4、属性情報を格納する属性テーブル5、パスIDと文字列とを対応付けたパスIDテーブル6、ラベルIDと文字列とを対応付けたラベルIDテーブル7を設け、木構造で表現されたXMLデータをノード単位で分割し、上記テーブル2～7に各ノードとリンク情報を関係付けて格納する。XMLデータを検索するには、問い合わせ処理手段9に対し問い合わせ文により問い合わせを行う。問い合わせ処理手段9は、インデックス8を用いて木構造を辿る問い合わせを実行し、要求された検索結果を出力する。

本発明の基本構成図



## 【特許請求の範囲】

【請求項1】 XMLで記述されたデータを、エレメントを中間ノードとし、エレメント値と属性値を葉ノードとし、タグをリンクとする木構造で表現し、XMLの木構造をノードとリンクに分解し、各ノードとリンク情報を関係付けて関係データベースのテーブルに格納し、

上記関係データベースに格納されたテーブルを利用して、任意の構造のXMLデータを検索することを特徴とするXMLデータの格納／検索方法。

【請求項2】 エレメントを中間ノードとし、エレメント値と属性値を葉ノードとし、タグをリンクとする木構造で表現されるXMLで記述されたデータを検索するシステムであって、

上記システムは、XMLデータを格納する格納手段を備え、該格納手段の関係データベースに、少なくとも中間ノードの情報を格納するための中間ノードテーブルと、リンクの情報を格納するためのリンクテーブルと、葉ノードの情報を格納するための葉ノードテーブルとを設け、

上記XMLの木構造をノードとリンクに分解して、上記テーブルに各ノードとリンク情報を関係付けて格納し、上記テーブルを参照して木構造を辿る問い合わせを実行し、XMLデータを検索することを特徴とするXMLデータ検索システム。

【請求項3】 関係データベースに、パスの文字列とパス用のIDの対応表であるパスIDテーブルと、ラベルの文字列とラベル用IDの対応表であるラベルIDテーブルとを設けたことを特徴とする請求項2のXMLデータ検索システム。

【請求項4】 リンクテーブルの中に各子エレメントがそのエレメント内で出現した順序の情報を付加し、葉ノードテーブルの中に各エレメント値がそのエレメント内で出現した順序の情報を付加し、上記情報により元のXML文書の復元を可能としたことを特徴とする請求項2または請求項3のXMLデータ検索システム。

【請求項5】 中間ノードテーブルに、ノードIDによる検索を高速に行なうためのインデックスと、テーブルの文書IDによる検索を高速に行なうためのインデックスと、パスIDによる検索を高速に行なうためのインデックスを用意し、

リンクテーブルに、親ノードから子ノードを高速に検索するためのインデックスと、子ノードから親ノードを高速に検索するためのインデックスを用意し、

葉ノードテーブルに、ノードIDからそのノードの値を得るためのインデックスと、ある値を持つノードを検索するためのインデックスを用意し、

パスIDテーブルに、パスの文字列に対応するパスIDを検索するためのインデックスを用意し、

ラベルIDテーブルに、ラベルの文字列に対応するラベ

ルIDを検索するためのインデックスを用意し、

上記インデックスを用いて木構造を辿る問い合わせを実行することを特徴とする請求項2、3または請求項4のXMLデータ検索システム。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】本発明は、XMLで記述された大量のデータを関係データベースに格納し、検索するXMLデータの格納／検索方法および検索システムに関し、特に、XML文書の構造に依存せずにあらゆるXMLデータを格納できるようにし、また格納されたXMLデータに対するXMLの木構造を辿る問い合わせを高速に実行できるようにしたXMLデータの格納／検索方法および検索システムに関するものである。

## 【0002】

【従来の技術】現在、XMLデータを格納するのに用いられている手法は、大まかに次の2つのタイプに分類することができる。

①ファイル格納：XML文書をファイル形式のまま格納する手法。この手法は、オリジナルのXMLファイルの全体あるいは一部をそのまま利用することを目的としており、そのため、XML文書をファイル形式のまま格納する。しかし、それだけでは、ファイルの数が増えたときに目的とするファイルを見つけ出すことが困難になるので、目的とするファイルを検索する為のインデックスも用意しておく必要がある。

【0003】②テーブル格納：XMLを関係データベースのテーブルにマッピングして格納する手法。この手法ではXML文書を構造化データと見なし、データベースに格納することによって高速な検索を行なうことを目的としている。そのため、この手法では、各エレメントを関係データベースのテーブルの各カラムにマッピングして格納する。XMLデータをテーブルにマッピングする為には、XMLの各エレメントをテーブルの各カラムにどのようにマッピングするかというマッピング規則が必要である。このマッピング規則はユーザが事前に指定する必要がある。

## 【0004】

【発明が解決しようとする課題】XMLデータを格納する際に一番問題となるのは、そのデータ構造が一意に定まっていないという事である。特に、DTD（文書型宣言）のないXMLデータでは、どこにどのようなタグが出現するか分からず、データ構造は全く分からない。DTDのあるXMLデータでさえも、DTDの中でタグの繰り返しやタグの選択、タグの再帰的な宣言が許されているので、データ構造が一意に定まらない。なお、このようなデータを半構造化データと呼ぶ。このようなデータ構造の定まっていないXMLデータを格納しようとする、格納スキーマの設計が問題となる。例えば、図8に示される〔DTD〕を持つ、サンプルXMLデータ〔X

MLデータ]をテーブル格納でデータベースに格納した場合を考える。なお、このサンプルXMLデータは、2冊の本の情報を含む書籍目録のデータである。

【0005】図9は上記XMLデータをテーブルに格納した様子を示す図である。図9のテーブルでは、1タブ  
ルが本1冊分の情報に相当していて、列にはXMLデータ  
中で出現する可能性のある全てのタグがとられてい  
る。これを見ると、一見サンプルデータが問題なく格納  
されているかのように見える。しかし、サンプルデータ  
のDTDに書かれた定義には著者数の制限が無いのに、  
図9のテーブルでは著者を格納するスペースは最大2人  
分しか用意されていない。もしXMLデータの中に著者  
がそれ以上存在したら、そのデータは格納できないか、  
格納しても情報が一部欠損することになる。このよう  
に、テーブル格納では、XMLのDTDで記述される繰  
り返しタグを格納することができない。これは、テー  
ブル格納ではあらかじめ格納する要素を列として指定し  
ておく必要があるため、最大数が未定の繰り返し要素を  
表現できないからである。また、同じ理由で再帰的に定義  
されているタグも格納できない。さらに、そもそもXML  
データにDTDが存在しなくて、どのようなタグが出現  
するか分かっていないときには、テーブルの構造を決  
められず、全く対応できない。

【0006】一方、ファイル格納は、XMLデータをフ  
ァイル形式のまま格納するので、DTDの無いXMLデ  
ータであろうと半構造のXMLデータであろうと、格納  
できないXMLデータは存在しない。しかし、それだけ  
では大量に格納されたデータの中から自分の求める情報  
だけを検索することができないので、検索用のインデッ  
クスが必要となる。インデックスの構成は目的に応じて  
色々と考えられ、簡単なものではタグ名と文字列の組を  
キーにして、そのタグに囲まれてその文字列が出現して  
いるようなXML文書を検索してくるというものがある。  
しかし、そのような簡単なインデックスでは、タグ  
の階層構造を考慮した検索は行なえない。タグの階層構  
造の情報を持つようにインデックスを工夫することも考  
えられるが、それでもなお次のことが問題として残る。

【0007】① インデックスがXMLの木構造の全  
ての情報を持っていないので、XMLデータの全情報を使  
った検索ができない。

② インデックスが木構造を辿ることに最適化されて  
いないので、そのような検索を行なった場合は検索速度が  
遅い。

以上のように、データ構造が一意に定まっていないXML  
データにおいては、いかにしてDTD無しのXMLデ  
ータや半構造のXMLデータを格納するか、また、格納  
されたXMLデータに対していかにして木構造を辿るよ  
うな複雑な問い合わせを高速に実行できるようにするか  
といった問題がある。本発明は上記した事情に鑑みなさ  
れたものであって、本発明の目的は、データ構造が一意

に定まっていないXMLデータをデータベースに格納  
し、複雑な問い合わせを高速に実行することができるXML  
データの格納/検索方法およびXMLデータ検索シ  
ステムを提供することである。

【0008】

【課題を解決するための手段】図1は本発明の基本構成  
を示す図である。同図に示すように、本発明のシステム  
は、エレメントを中間ノードとし、エレメント値と属性  
値を葉ノードとし、タグをリンクとする木構造で表現さ  
れるXMLで記述されたデータを検索するシステムにお  
いて、XMLデータを格納する格納手段1を設け、該格  
納手段1の関係データベースに、少なくとも中間ノード  
の情報を格納するための中間ノードテーブル2と、リン  
クの情報を格納するためのリンクテーブル3と、葉ノ  
ードの情報を格納するための葉ノードテーブル4とを設け  
る。そして、上記XMLの木構造で表現されたXMLデ  
ータをノード単位で分割し、上記テーブル2〜4に各ノ  
ードとリンク情報を関係付けて格納する。XMLでは、  
木構造を形成する中間ノードと、エレメントの値を持  
っている葉ノードとでは、格納するために最適な格納構造  
が異なるので、上記のようにそれぞれ最適化された別々  
の専用テーブルに格納するのが望ましい。このように、  
値を持つためのノードである葉ノードと木構造の情報を  
持つためのノードである中間ノードを別々のテーブルに  
格納することにより、値を格納するための格納スペース  
を節約することが可能となる。各ノード間の接続情報を  
保持する為のリンクも、リンクテーブル3に格納して持  
っておく必要がある。また、属性情報を格納するための  
属性テーブル5を別途設けてもよい。さらに、中間ノ  
ードテーブル2に各ノードのルートからのフルパス情報を  
IDで記述し、パス用のIDと文字列の対応表をパスID  
テーブル6として別に持つことにより、格納スペース  
の節約と、検索の高速化を図ることができる。同様に、  
リンクテーブル3のタグ名と属性ノードテーブルの属性  
名をIDで記述し、これらラベルのIDと文字列の対応  
表をラベルIDテーブル7として別に持つことによ  
って、格納スペースの節約と文字列検索の高速化を図る  
ことができる。また、リンクテーブル3の中に各子エレ  
メントがそのエレメント内で出現した順序の情報を付加  
し、葉ノードテーブルの中に各エレメント値がそのエレ  
メント内で出現した順序の情報を付加することにより、  
元のXML文書の復元が可能となる。

【0009】本発明では、XMLの木構造をそのまま格  
納手段1に格納するので、DTD無しのXMLデータや  
半構造のXMLデータも格納できる。また、XMLの木  
構造を全てデータベース上に格納しているので、木構造  
の全ての情報を検索に利用することができる。しかしこ  
れだけでは問い合わせが行なわれたときに、ノード単位  
に分割して格納されているXMLデータの木構造を再結  
合するのに時間がかかり、問い合わせの実行時間が遅く

なる。そこで本発明では、上記のテーブル2〜7に、XMLデータへの問い合わせパターンを考慮してインデックス8を張る。これにより、XMLの木構造を辿るような複雑な問い合わせの実行を高速に行なうことを可能となる。上記XMLデータを検索するには、例えばXMLデータ検索言語により、問い合わせを行う。これにより問い合わせ処理手段9は、問い合わせ文の構文チェックを行い問い合わせのための構文木を生成し、最適な実行プランを生成する。この実行プランは、木構造検索用の関数セットで記述される。この実行プランにより、上記インデックス8を用いて木構造を辿る問い合わせを実行し、要求された検索結果を出力する。

【0010】本発明においては、次のように構成することもできる。

(1) テーブルに関係データベースの制約の機能を適用することによって、XMLの構文規則をチェックする。

(2) リンクテーブルの中に、各要素の同ラベルを持つ兄弟要素中での出現順序の情報を付加し、各ラベルの出現順序を指定した問い合わせの実行を可能とする。

(3) リンクテーブルにリンクの両節点の情報だけでなくタグ名の情報も持つことによって、タグ名を指定してリンクを辿る問い合わせを高速に実行する。

(4) 属性テーブルの中の属性ノードの接続先をリンクではなくて中間ノードにすることによって、属性を条件にして木構造を辿る問い合わせを実行する際のテーブル検索回数を削減し、問い合わせの高速実行を可能とする。

(5) 中間ノードテーブルのパスIDによる検索を高速に行なうためのインデックスをB+-treeで構築する場合において、キー値をパスIDとノードIDの組とすることによってキー値の重複を無くす。

(6) 中間ノードテーブルの文書IDによる検索を高速に行なうためのインデックスをB+-treeで構築する場合において、キー値を文書IDとノードIDの組とすることによってキー値の重複を無くす。

【0011】

【発明の実施の形態】以下、本発明の実施の形態について説明する。

#### (1) システム構成

図2は本発明の実施例のシステムの構成を示す図である。同図に示すように、本実施例のシステムは大きくわけて、XMLデータ格納部11、XMLデータ格納部11にXMLデータを挿入するためのXMLデータ挿入モジュール12、格納されたXMLデータへの問い合わせを処理する問い合わせ処理エンジン部13から構成される。XMLデータは、XMLデータ挿入モジュール12によって、XMLデータ格納部11に挿入される。XMLデータ挿入モジュール12は、XMLパーザ12aとローダー12bから成り、XMLパーザ12aは入力さ

れたXMLデータを構文解析し、XMLデータの木構造を、XMLデータ格納部11に格納できるようにノード単位に分解する。また、ローダー12bは、そのノード単位に分解された木構造をXMLデータ格納部11のテーブルに挿入する。

【0012】図3に上記XMLデータの格納処理を示すフローチャートを示す。本実施例においてXMLデータの格納処理は次のように行われる。まず、ステップS1において、XMLファイルを読み込む。ステップS2において、XMLパーザにより、入力ファイルの構文解析を行う。解析が成功した場合には、ステップS3に行き、XMLパーザが解析結果として、XMLの木構造のノード情報とリンク情報を中間形式としてファイル出力する。また、解析が成功しない場合には、構文解析失敗としてエラー出力し処理を終了する。ステップS4において、生成された中間形式ファイルを読み込み、ステップS5において、読み込んだXMLデータをローダによって関係データベースの各テーブルに挿入し、処理を終了する。また、上記挿入が成功しない場合には、データ挿入失敗としてエラー出力をして処理を終了する。

【0013】格納されたXMLデータに対する問い合わせは、XMLデータ問い合わせ言語で行なわれ、その問い合わせは問い合わせ処理エンジン13で処理される。問い合わせ処理エンジン13は、問い合わせ言語のパーザ13a、問い合わせ最適化エンジン13b、木構造検索用API（アプリケーション・プログラミング・インタフェース）13cから成る。問い合わせ言語のパーザ13aは、入力された問い合わせ文の構文チェックを行い問い合わせのための構文木を生成する。問い合わせ最適化エンジン13bは、上記構文木を基に、最適な実行プランを生成する。この実行プランは、木構造検索用API13cの関数セットで記述される。木構造検索用API13cは、XMLデータ格納部11とのインタフェースで、XMLの木構造上での基本的な検索を行なう関数のセットである。

【0014】次に、上記システムにおける各部の構成についてさらに詳細に説明する。

#### (1) テーブル構成

まず、上記XMLデータ格納部11に格納されるテーブルの構成について説明する。XMLデータを木構造で表現する方法はいくつかあるが、本実施例では図4に示す木構造表現を想定している。図4は、前記図8に示したXMLデータを木構造で表現したものである。この木構造表現において、丸い中間ノードはエレメントを表しており、ノードの親子関係がエレメントの包含関係を表している。

【0015】また、ノードの丸の中の数字はノードIDを表している。ノードとノードを結ぶリンク（枝）はタグを表しており、リンクの横に書かれている文字列はタグ名を表している。三角の葉ノードはエレメントの値を

表し、四角い葉ノードはタグに付けられた属性(Attribute)を表している。値を持つのはこの2つの葉ノードだけである。ノードを分割してデータベースに格納するときに、ノードの情報だけをデータベースのテーブルに格納したのでは、木構造のノード間の繋がり、つまりリンクの情報が欠落してしまう。そこで、リンクの情報はリンクの情報としてそれを格納する専用のテーブルを用意する。またノードも、中間ノードと、エレメント値の葉ノード、属性の葉ノードとは最適な格納構造が異なるので、別々のテーブルに格納する必要がある。

【0016】本実施例で使用するテーブルは、全部で次の6つである。

#### ①中間ノードテーブル

これは中間ノードの情報を格納するテーブルである。ノードID(id)の他に、そのノードが含まれている文書の文書ID(docid)、そのノードまでのルートからのフルパスのID(pathid)をカラムとして持っている。

#### ②リンクテーブル

これはノード間のリンクを格納するテーブルである。ノードID(id)、リンクのラベル(タグ名)のID(labelid)、子ノードのノードID(child)、その子ノードの全兄弟ノード中での出現順序(tord:total order)、その子ノードの同ラベルを持つ兄弟ノード中での出現順序(pord:partial order)をカラムとして持っている。上記のように、リンクテーブル中にラベル(タグ名)のID(labelid)を付加することによりタグ名を指定してリンクを辿る問い合わせを高速に実行することが可能となる。

#### 【0017】③葉ノードテーブル

これはエレメント値の葉ノードを格納するテーブルである。そのエレメントにあたる中間ノードのノードID(id)の他に、エレメントの値(value)と、そのエレメント中でその値が出現した順序(order)をカラムとして持っている。このように、値を持つための葉ノードテーブルを、前記中間ノードテーブルとは別に設けることにより、値を格納するスペースを節約することができる。

#### 【0018】④属性ノードテーブル

これはタグにつけられた属性(例えば図8における<book year="1995">におけるyear)を格納するテーブルである。そのタグが含まれるエレメントにあたる中間ノードのノードID(id)の他に、属性名のID(labelid)、属性値(Attvalue)をカラムとして持つ。なお、属性テーブルに関係データベースの制約機能を用いて、(id,labelid)の組がユニークという制約をかけておくことにより、「同一のタグ内では同一の属性名は出現してはならない」というXMLの属性に関する構文規則をチェックすることができる。また、本実施例で想定している木構造表現では、XMLのタグが木構造のリンクに相当するので、XMLのタグに付けられる属性は本来ならばリンクに付くべきである。しかし、図4では、属性はリンクに対してではなく、その下のノードに付いている。これ

は、検索時のテーブル参照の回数を少なくするためである。すなわち、属性を条件として木構造を辿る問い合わせを実行する際のテーブル検索回数を削減し、問い合わせの高速化を図ることが可能となる。

#### 【0019】⑤パスIDテーブル

これはパスIDとパスの文字列の対応表である。パスの文字列を中間ノードテーブルに直接書き込まないでこのように別に持っているのは、スペースの節約の為もあるが、パス名の文字列マッチングを含む検索が行なわれたときに、検索対象が少なくてすみ、検索が高速化できるからでもある。

#### ⑥ラベルIDテーブル

これはラベルIDとラベルの文字列の対応表である。このように、リンクテーブルのタグ名と、属性ノードテーブルの属性名をIDで記述し、このラベルのIDと文字列の対応表をラベルIDテーブルとして別に持つことにより、パスIDテーブルと同様、格納スペースの節約と、検索の高速化を図ることができる。

【0020】また、上記のように、リンクテーブル中に、子ノードの全兄弟ノード中での出現順序(tord:total order)の情報を付加し、また、葉ノードテーブル中に、各エレメント値がそのエレメント内で出現した順序(order)の情報を付加することにより、XMLデータ格納部11に格納されるノード単位に分解されたXMLデータから、元のXML文書を復元することが可能となる。例えば、「今日は<天気> 晴れ</天気> だった。〇〇は<場所> デパート</場所> へでかけた。」のようにタグで区切られた文章を復元することも可能になる。また、リンクテーブル中に、各エレメントの同ラベルを持つ兄弟ノード中での出現順序(pord:partial order)の情報を付加することにより、各ラベルの出現順序を指定した問い合わせを高速に実行することが可能となる。

【0021】一例として、図8のサンプルXMLデータ(図4の木構造表現)を上記のテーブル群で格納した様子を図5、図6に示す。図5は中間ノードテーブル、リンクテーブルの例を示す図である。中間ノードテーブルにおいて、例えば、第1行目のid(=5)は図4において"5"と記されたノードを示し、そのノードが含まれている文書の文書ID(docid)は1である。また、そのノードまでのルートからのフルパスのID(pathid)は1であり、このIDに対応したpathは、"bib.book.publisher.name"である。また、リンクテーブルにおいて、例えば1行目のid(=4)は図4において、"4"と記されたノードを示し、そのlabelidは5であり、このlabelidに対応するlabelは"name"である。また、その出現順序を示すtord,pordはそれぞれ"0","0"であり、子ノードは、図4で"5"と記されたノードである。

【0022】図6は葉ノードテーブル、属性ノードテーブル、パスIDテーブル、ラベルIDテーブルの例を示す図である。葉ノードテーブルにおいて、例えば第1行

目のid (=5) は図4において、"5" と記されたノードを示し、そのorder は"0"、またその葉ノードの値(value)は"Addison-Wesley"である。属性ノードテーブルにおいて、例えば第1行目のid (=3) は図4において、"3" と記されたノードを示し、そのlabelid は3 ("year"に対応)、その属性値(attvalue)は"1995"である。また、パスIDテーブル、ラベルIDテーブルにはそれぞれ、上記各テーブル中のpathid、labelid に対応したパスの文字列、ラベルの文字列が格納され、例えば、pathid="1" に対応した文字列は前記したように"bib.book.publisher.name"であり、また、例えばlabelid="1" に対応した文字列は"bib"である。

#### 【0023】(2) インデックスの構成

本実施例においては、本来連結されていたはずの木構造のノードが、前記したように1つ1つに分割されて関係データベースのテーブルに格納されている。このために、木構造を辿る問い合わせが行なわれた場合、問い合わせで辿る部分のリンクを連結し直すためにジョイン操作が行なわれる。このジョイン操作の速度は全体の検索速度に大きく影響するので、ジョイン操作を高速に行なえるようにインデックスを効果的に張っておく必要がある。また、問い合わせが行なわれる場合、検索条件として指定されるのは、エレメントの値、属性、パス、出現順序などである。それらの検索も高速に行なう必要があるため、そこにもインデックスを用意しておく必要がある。

【0024】図7に、上記図5、図6に示したテーブルに張ったインデックスの一覧を示す。このインデックスはB+-treeで張っており、キーが複数の属性の組からなるインデックスは、その組の先頭からの部分的な属性の組で検索に用いることもできる。なお中間ノードテーブルに張ってあるインデックスでキーが(pathid,id)のものは、あるパスに該当する全てのノードを検索してくるときに使用するものである。このインデックスのキーは、一見pathid単独で構わないように思われるかもしれない。しかしキーをpathidだけにすると、同じキー値を持つエントリが多量に発生して、B+-tree インデックスが機能しなくなる。上記のようにキー値をパスID(pathid)とノードのID(id)の組とすることにより、キー値の重複を無くすことができ、B+-tree の検索を高速に行うことができる。また、中間ノードテーブルに張ってあるインデックスでキーが(docid,id)も同様であり、文書ID(docid)とノードのID(id)の組とすることにより、キー値の重複を無くすことができ、B+-tree の検索を高速に行うことができる。

#### 【0025】(3) 問い合わせの実行

前記したように、格納されたXMLデータに対する問い合わせは、例えばXMLデータの問い合わせ言語で行なわれる。XMLデータのための検索言語の一つとして検

索言語XQLがある。XQLによる問い合わせ文を、例により簡単に説明する。

【0026】

```
SELECT result:<$book.title>
```

```
FROM book: bib.book
```

```
WHERE $book.author.lastname="Darwen";
```

この問い合わせの意味は「bib.book.author.lastnameがDarwenであるようなbib.bookについて、bib.book.titleを検索結果として得たい」という意味である。

【0027】上記に示すように、問い合わせ文は大きく、SELECT、FROM、WHERE の3つの部分に別れている。SELECTの部分では検索結果として得たいエレメントのプロジェクションを指定する。FROMの部分では検索の対象となるエレメントを指定している。WHEREの部分では検索の条件のセレクションを指定する。上記のような問い合わせは前記したように、問い合わせ処理エンジン13で処理される。問い合わせ処理エンジン13では、上記のような問い合わせ文の構文チェックを行い問い合わせのための構文木を生成する。そして、該構文木を基に、最適な実行プランを生成する。この実行プランは、木構造検索用の関数セットで記述される。

【0028】次に、上記XMLデータに対する問い合わせ処理が、どのように行なわれるかを説明する。ここでは、図8のサンプルXMLデータを、XMLデータ格納部11に格納し、前述した図5、図6に示したテーブルに挿入した場合を例として、上記のように「著者がDarwenである本のタイトルを求めよ」という問い合わせを行なった場合について説明する。この場合のテーブル検索は、次のように行われる。なお、下記1. ~10. の処理は、上記木構造検索用の関数により実行される。

- 【0029】1. 葉ノードテーブルを検索して、値が"Darwen"であるノードのノードID (=16) を得る。
2. パスIDテーブルを検索して、パス"bib.book.author.lastname" のパスID (=4) を得る。
3. 中間ノードテーブルを上記1. で得られたノードID (=16) で検索して、得られたパスID (=4) が上記2. で得られたパスID (=4) と一致することを確認する。
4. ラベルIDテーブルを検索して、ラベル"lastname" のラベルID (=8) を得る。
5. リンクテーブルを検索して、上記1. で得られたノードID (=16) と上記4. で得られたラベルID (=8) から、親ノードのノードID (=15) を得る。
6. ラベルIDテーブルを検索して、ラベル"author" のラベルID (=7) を得る。
7. リンクテーブルを検索して、上記5. で得られたノードID (=15) と上記6. で得られたラベルID (=7) から、親ノードのノードID (=9) を得る。



11

8. ラベルIDテーブルを検索して、ラベル"title"のラベルID(=6)を得る。

9. リンクテーブルを検索して上記7. で得られたノードID(=9)と上記8. で得られたラベルID(=6)から、子ノードのノードID(=12)を得る。

10. 葉ノードテーブルを検索して、上記9. で得られたノードID(=12)から、そのノードの値("Foundation for Object/Relational Database")を得る。以上のようにして得られた検索結果は、問い合わせ処理エンジン13を介して出力され、ユーザに提示される。

【0030】

【発明の効果】以上説明したように、本発明においては、関係データベースに、中間ノードの情報を格納するための中間ノードテーブルと、リンクの情報を格納するためのリンクテーブルと、葉ノードの情報を格納するための葉ノードテーブル等のテーブルを設け、XMLの木構造をノードとリンクに分解して、上記テーブルに各ノードとリンク情報を関係付けて格納し、上記テーブルを参照して木構造を辿る問い合わせを実行し、XMLデータを検索するようにしたので、データ構造が一意に定ま

っていないXMLデータに対する複雑な問い合わせを高速に実行することができる。また、XMLの木構造をそのまま格納手段に格納するので、DTD無しのXMLデータや半構造のXMLデータも格納することができる。さらにXMLの木構造を全てデータベース上に格納している

ので、木構造の全ての情報を検索に利用することができる。

【図面の簡単な説明】

【図1】本発明の基本構成図である。

【図2】本発明の実施例のシステムの構成例を示す図である。

12

【図3】本発明の実施例のシステムにおける格納処理フローを示す図である。

【図4】XMLデータの木構造表現の一例を示す図である。

【図5】本発明の実施例のテーブル構成の一例を示す図(1)である。

【図6】本発明の実施例のテーブル構成の一例を示す図(2)である。

【図7】本発明の実施例のインデックス一覧を示す図である。

【図8】XMLデータの一例を示す図である。

【図9】図8のXMLデータをテーブルに格納した様子

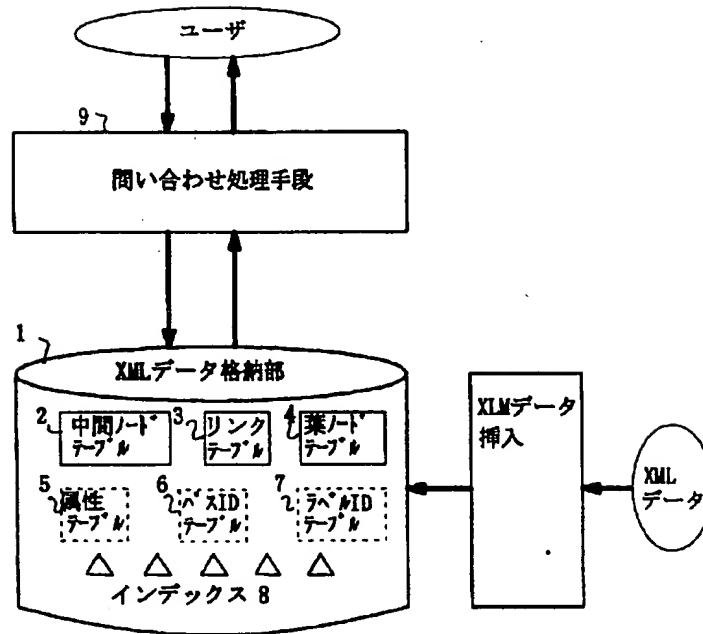
を示す図である。

【符号の説明】

- 1 XMLデータ格納格納手段
- 2 中間ノードテーブル
- 3 リンクテーブル
- 4 葉ノードテーブル
- 5 属性テーブル
- 6 バスIDテーブル
- 7 ラベルIDテーブル
- 8 インデックス
- 9 問い合わせ処理手段
- 11 XMLデータ格納部
- 12 XMLデータ挿入モジュール
- 12a XMLパーザ
- 12b ロード
- 13 問い合わせ処理エンジン部
- 13a 問い合わせ言語のパーザ
- 13b 問い合わせ最適化エンジン
- 13c 木構造検索用

【図1】

## 本発明の基本構成図



【図7】

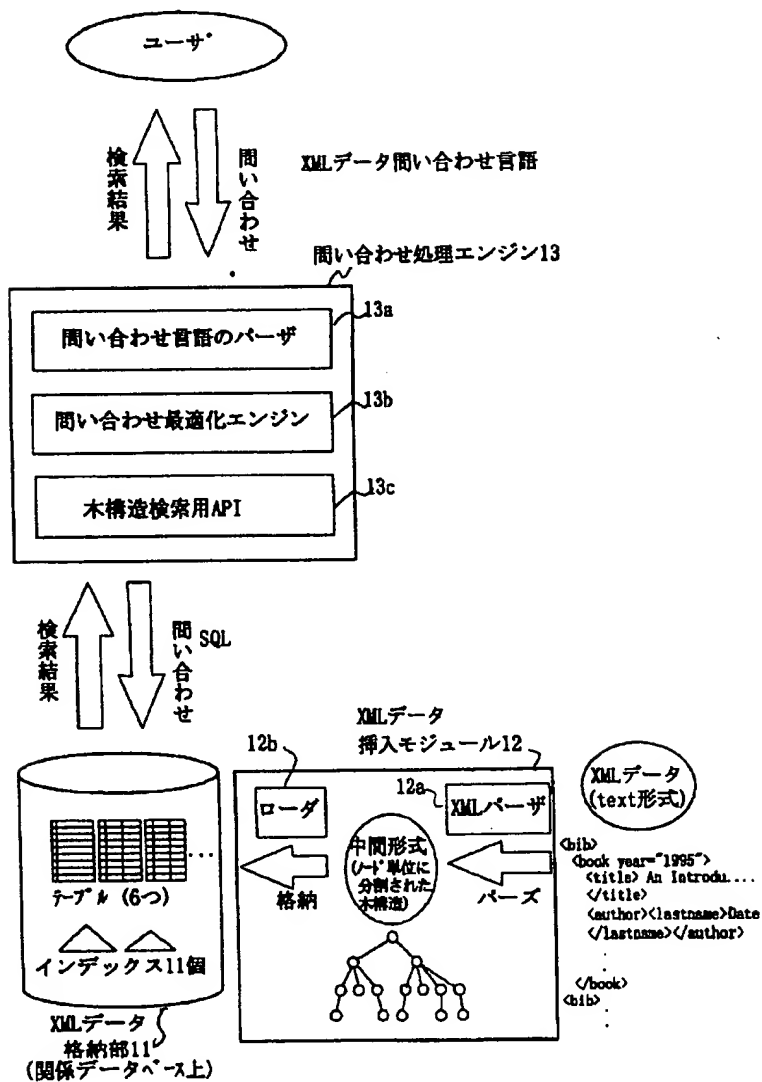
## 本発明の実施例のインデックス一覧を示す図

## インデックス一覧

テーブル名	キー
中間ノード	id
中間ノード	(docid, id)
中間ノード	(pathid, id)
リンク	(id, labelid, child)
リンク	(child, labelid, id)
葉ノード	id
葉ノード	value
属性ノード	id
属性ノード	attvalue
パスID	path
ラベルID	label

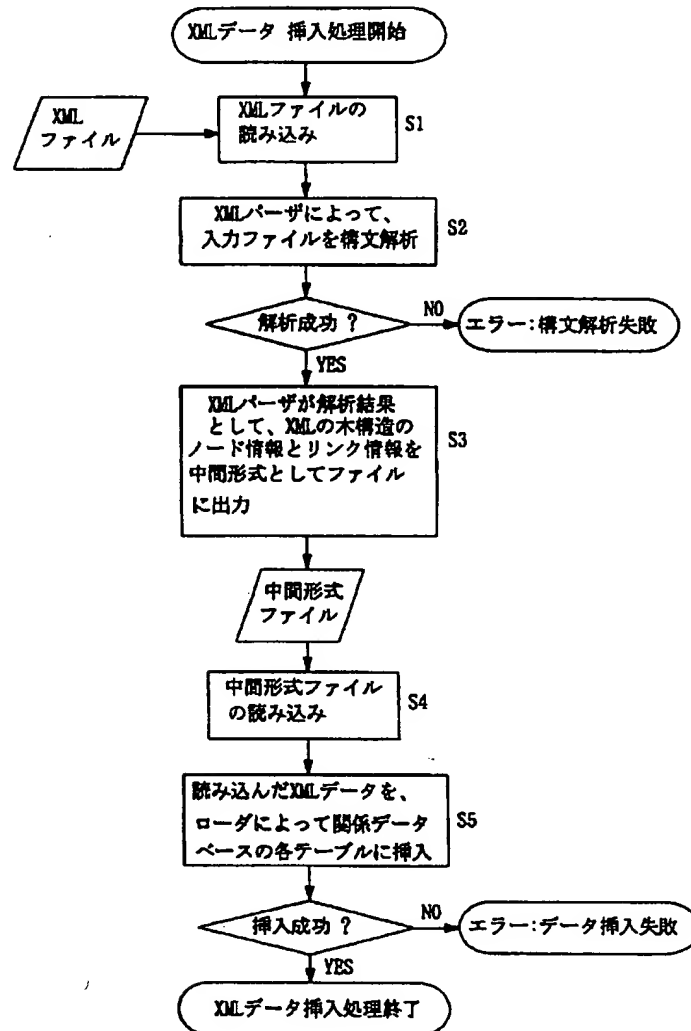
【図2】

本発明の実施例のシステムの構成例を示す図



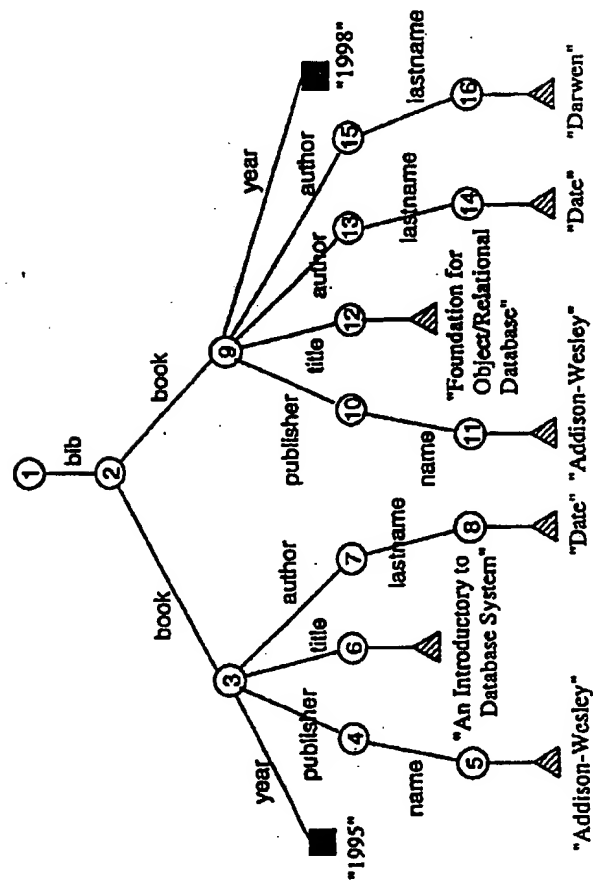
【図3】

本発明の実施例のシステムにおける格納処理フローを示す図



【図4】

XMLデータの木構造表現の一例を示す図



【図5】

本発明の実施例のテーブル構成の一例を示す図(1)

中間ノードテーブル

id	docid	pathid
5	1	1
4	2	2
6	1	3
8	1	4
7	1	5
3	1	6
11	1	1
10	1	2
12	1	3
14	1	4
13	1	5
16	1	4
15	1	5
9	1	6
2	1	7
1	1	8

リンクテーブル

id	labelid	tord	pord	child
4	5	0	0	5
7	8	0	0	8
3	4	0	0	4
3	6	1	0	6
3	7	2	0	7
10	5	0	0	11
13	8	0	0	14
15	8	0	0	16
9	4	0	0	10
9	6	1	0	12
9	7	2	0	13
9	7	3	1	15
2	2	0	0	3
2	2	1	1	9
1	1	0	0	2

【図6】

本発明の実施例のテーブル構成の一例を示す図(2)

葉ノードテーブル

id	order	value
5	0	Addison-Wesley
6	0	An Introductory to Database System
8	0	Date
11	0	Addison-Wesley
12	0	Foundation for Object/Relational Database
14	0	Date
16	0	Darwen

属性ノードテーブル

id	labelid	attvalue
3	3	1995
9	3	1998

パスIDテーブル

pathid	path
1	bib. book. publisher. name
2	bib. book. publisher
3	bib. book. title
4	bib. book. author. lastname
5	bib. book. author
6	bib. book
7	bib
8	/

ラベルIDテーブル

labelid	label
1	bib
2	book
3	year
4	publisher
5	name
6	title
7	a author
8	lastname

【図8】

## XMLデータの一例を示す図

## [ DTD ]

```

<ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA>
<ELEMENT article (author+, title, year?, (shortversion | longversion))>
<!ATTLIST article type CDATA>
<ELEMENT publisher (name, address?)>
<ELEMENT author (firstname?, lastname?)>

```

## [ XMLデータ ]

```

<bib>
  <book year="1995">
    <title> An Introductory to Database System </title>
    <author> <lastname> Date </lastname> </author>
    <publisher> <name> Addison-Wesley </name> </publisher>
  </book>
  <book year="1998">
    <title> Foundation for Object/Relational Database </title>
    <author> <lastname> Date </lastname> </author>
    <author> <lastname> Darwen </lastname> </author>
    <publisher> <name> Addison-Wesley </name> </publisher>
  </book>
</bib>

```

【図9】

## 図8のXMLデータをテーブルに格納した様子を示す図

## bookのテーブル

ID	title	author1 firstname	author1 lastname
1	An Introductory to Database System		Date
2	Foundation for Object/Relational Database		Date
⋮	⋮	⋮	⋮

author2 firstname	author2 lastname	publisher name	publisher address	year
		Addison-Wesley		1995
	Darwen	Addison-Wesley		1998
⋮	⋮	⋮	⋮	⋮



フロントページの続き

(72)発明者 石川 博

Fターム(参考) 5B075 ND36 PP23 QR00 QT06

神奈川県川崎市中原区上小田中4丁目1番

1号 富士通株式会社内